小马哥的 Java 项目实战营

微服务项目 - 第二十六节 分布式配置

小马哥（mercyblitz）

我是谁？

小马哥（mercyblitz）

- 父亲
- Java 劝退师
- Apache Dubbo PMC
- Spring Cloud Alibaba 架构师
- 《Spring Boot 编程思想》作者

预习知识

- MicroProfile Config
- Spring Environment 抽象
- Spring 配置源（PropertySources）

议题

- Spring Cloud 配置基础
- Spring Cloud Config 架构设计
- Spring Cloud Alibaba Nacos Config 实现分布式配置
- Spring Cloud Alibaba Nacos Config 实现 Bean 的动态更新
- 问答互动

Spring Cloud 配置基础

- 基本概念

配置（Configuration）是应用程序重要的元信息（Metadata），几乎所有软件依赖配置调控程序行为，比如 MySQL 中的 my.ini 文件，Apache Maven 的配置文件 settings.xml 文件，以及其项目管理文件 pom.xml。尽管配置常以文件的形式承载，然而并仅限于此。对于应用程序而言，配置的内容远比存储介质重要，同时，应用也有偏好的配置格式，比如 Key-Value 格式、XML 格式、JSON 格式以及自定义格式等。

Spring Cloud 配置基础

- 配置分类
  - 本地配置（Local Configuration）
    - 内部配置（Internal Configuration）
    - 外部配置（External Configuration）
  - 远程配置（Remote Configuration）
    - 版本化配置（Versioned Configuration）
    - 分布式配置（Distributed Configuration）

Spring Cloud 配置基础

- 内部配置（Internal Configuration）

通常，内部配置是通过程序代码，甚至是硬编码（Hard Code）实现。虽然不推荐程序使用硬编码来初始化配置，然而这并非是不可原谅的做法，比如以下 Spring Bean 的配置：

```
@Bean
public Customizer<SentinelCircuitBreakerFactory> defaultConfig() {
    return factory -> {
        factory.configureDefault(
            id -> new SentinelConfigBuilder().resourceName(id)
            .rules(Collections.singletonList(new DegradeRule(id)
                                             .setGrade(RuleConstant.DEGRADE_GRADE_RT)
                                             .setCount(100)
                                             .setTimeWindow(10)))
            .build());
    };
}
```

Spring Cloud 配置基础

- 外部配置（External Configuration）

外部配置是软件使用者、开发者以及运维者最常见和熟悉的配置手段，正如前文提到的 MySQL my.ini 文件等。在 Java 生态体系中，尽管文件存储配置也是最通用的方式，然而 Java 作为一门主流并成熟编程语言，JDK 层面允许程序读取来自 Java System Properties、操作系统环境变量以及 JNDI 等多方配置来源，如前文在讨论的共享线程池时，makeCommonPool() 方法中部分实现被省略，而该部分实现则采用了 Java System Properties 作为外部配置源。

Spring Cloud 配置基础

- 远程配置（Remote Configuration）

在分布式场景中，远程配置的内容来自于配置运用端程序进程物理环境以外的环境，其作用相当重要，是大多数互联网企业的基础设施标配，大多数实现采用 C/S（客户端/服务器）架构。此处的 C（Client）并非狭隘地使用特定的客户端，可以泛指 Web 客户端或 HTTP Client，换言之，C/S 架构广义地包含了 B/S（浏览器/服务器）架构，这两种实现均采用 B/S 架构，属于“分布式配置（Distributed Configuration）B/S 架构，然而都归类于“分布式配置实现，即应用需要依赖一个专属的 HTTP 客户端，通过访问远端的 Web 服务器获取配置。实现上，配置客户端版本化配置（Versioned Configuration）”。

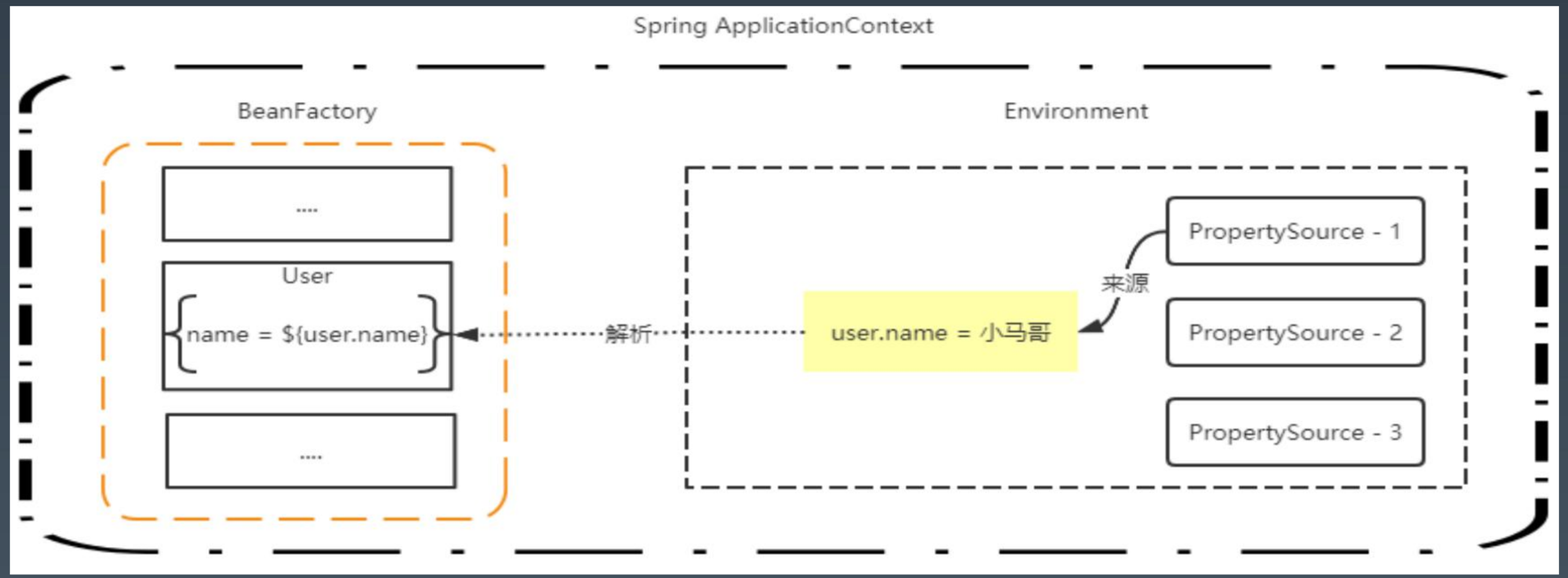
Spring Cloud 配置基础

- Spring 配置生命周期

在 Spring Framework 体系中，Spring ApplicationContext （应用上下文）是 Spring 应用中枢组件，用于管理 Spring IoC 容器（ BeanFactory ）、Bean 的生命周期，以及企业级特性，如 Spring 配置能处理占位符（Placeholder），用于 BeanFactory 和 Bean 的属性替换，其数据来源于 Spring 配置源 PropertySources，它由多个 PropertySource 组成，其中包含我们熟知 Java 系统属性（ System#getProperties()）以及 OS 环境变量（ Environment）。不过 PropertySources 对象通常不直接暴露给开发人员，而以 Spring ConfigurableEnvironment 对象作为门面，即通过 ConfigurableEnvironment#getPropertySources() 方法获取，Spring 配置示意架构如下图所示：

Spring Cloud 配置基础

- Spring 配置生命周期



Spring Cloud Config 架构设计

- 简介

架构上，Spring Cloud Config 属于 Spring Cloud 配置的重要组成部分，它由 Spring Cloud Config Client 和 Spring Cloud Config Server 组成，Spring Cloud Config Client 可视作一个具备“负载均衡”能力的 HTTP 配置读取客户端，在配置客户端（消费端）的使用方式相对固定，通过外部化配置来设置 Web 配置服务器的地址，而 Spring Cloud Config Server 可以基于不同的基础设施实现，前文提到的“版本化配置（Versioned Configuration）”就是其中一种，如 Spring Cloud 官方版本内建了 Spring Cloud Config Server 的 Git 和 SVN 实现，后续版本又新增了对 JDBC 以及 Redis 等支持，当然也包括三方整合，比如：AWS S3 和 HashiCorp Vault 等。

Spring Cloud Config 架构设计

- 内建实现

配置服务器技术	配置类型	配置来源	说明
Git	版本化配置	Git 仓库	服务器配置通过 JGit 技术访问 Git 仓库或本地文件系统获得
SVN	版本化配置	SVN 仓库	服务器配置通过 SvnKit 技术访问 SVN 仓库或本地文件系统获得
JDBC	分布式配置	兼容JDBC的关系型数据库	服务器配置通过 JDBC 连接关系型数据获得
Redis	分布式配置	Redis 服务	服务器配置通过 Redis Spring 访问 Redis 服务器获得
Nacos	分布式配置	Nacos 配置服务	服务器配置通过 Nacos Config SDK 访问 Nacos 配置服务器获得
CredHub	分布式配置	CredHub 服务器	服务器配置通过 SDK 访问 CredHub 服务器获得
AWS S3	分布式配置	AWS S3 服务	服务器配置 SDK 访问 AWS S3 服务器获得
HashiCorp Vault	分布式配置	Vault 服务器	服务器配置通过 HTTP 请求访问 Vault 服务器获得

Spring Cloud Config 架构设计

- 连接模式

Spring Cloud Config Client 连接 Spring Cloud Config Server 存在两种模式：“地址直连模式”和“服务发现模式”。连接模式仅作用于 Spring Cloud Config Client 一端。

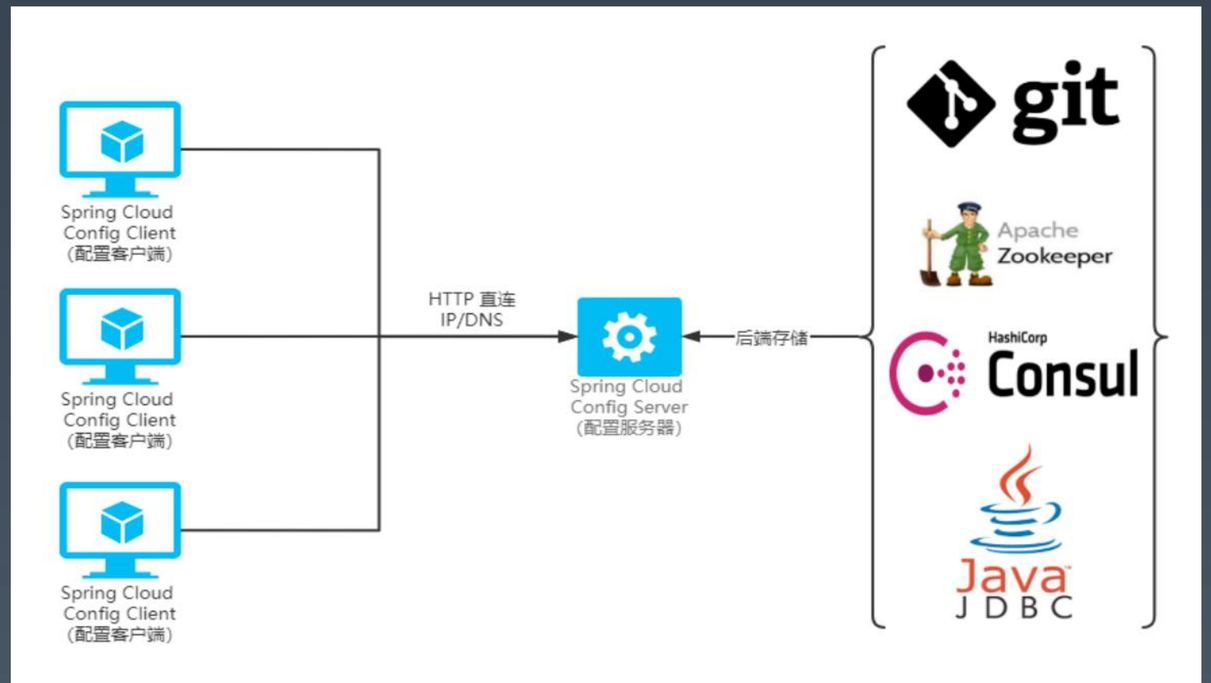
- 地址直连模式

- 服务发现模式

Spring Cloud Config 架构设计

- 地址直连模式

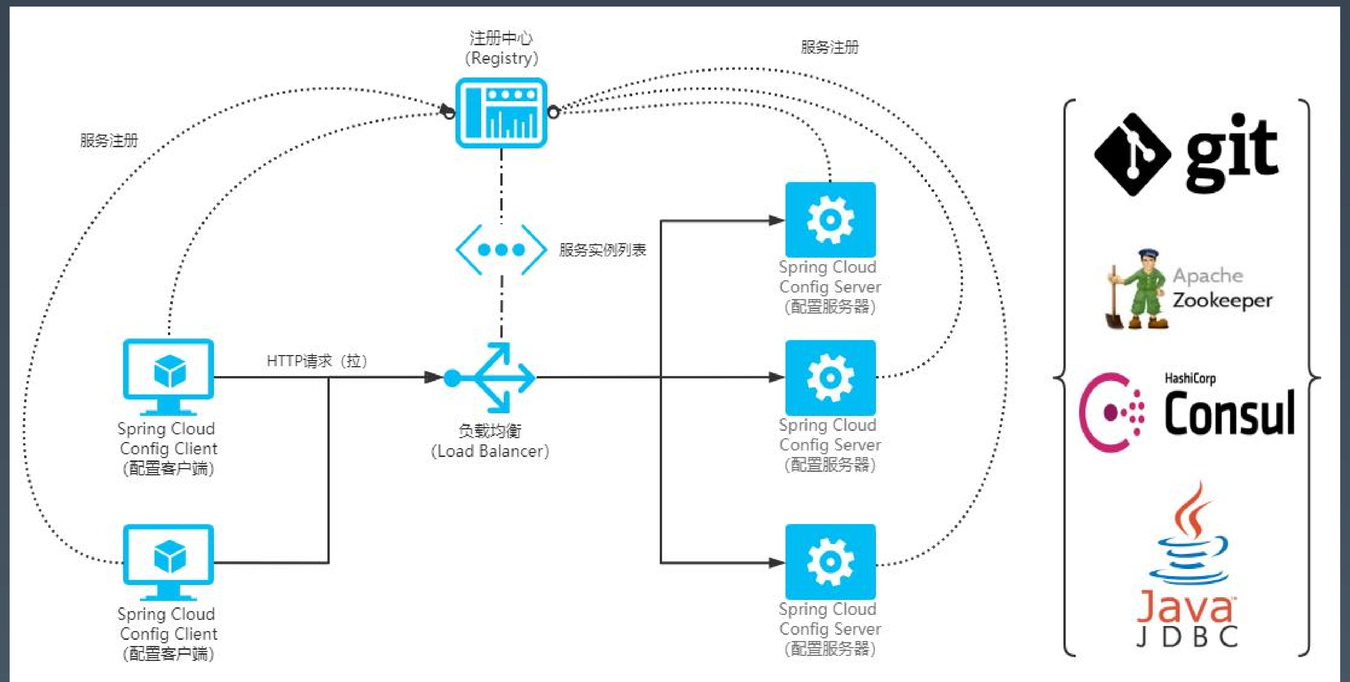
地址直连模式是 Spring Cloud Config Client 通过配置 spring.cloud.config.uri 直接连接 Spring Cloud Config Server 服务器地址的方式，此处的地址可以是服务器 IP、主机（hostname）或者 DNS，默认值为："http://localhost:8888"，具体架构如下所示：



Spring Cloud Config 架构设计

- 服务发现模式

Spring Cloud Config Client 服务发现模式实在“地址直连模式”相同的 HTTP 通讯模式基础上，增加“服务注册与发现（Service Registration and Discovery）”和“负载均衡（Load Balancing）”特性，具体架构如下所示：



THANKS!